Building a BSP Layer: An Overview of meta-tegra

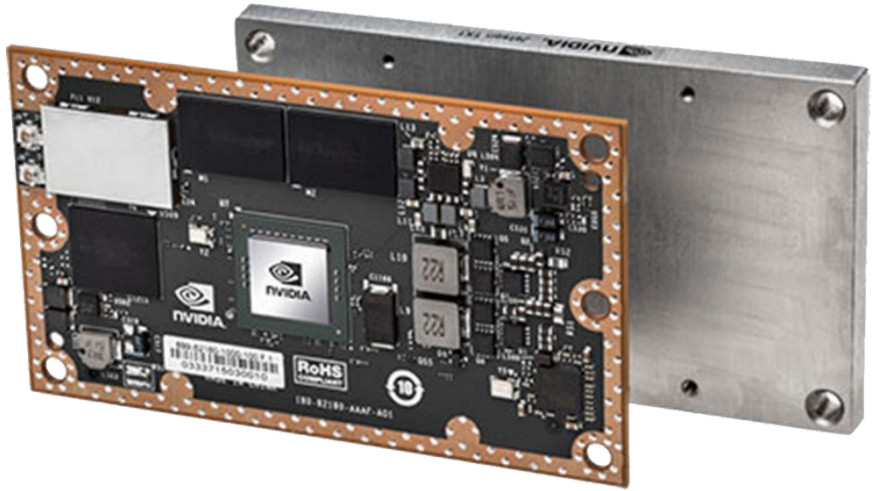
Matt Madison
Alcatraz AI, Inc.
matt@madison.systems

Yocto Project *Virtual* Summit Europe, October 29-30, 2020

Jetson Hardware

Visit https://developer.nvidia.com/embedded-computing
for more information

Jetson TX1

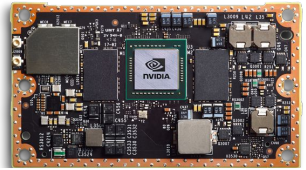


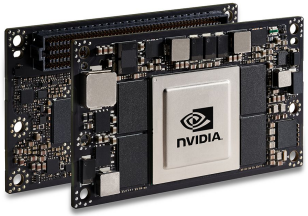
First of the 64-bit Jetsons, introduced in 2015

Dev kit now EOL (but compatible with TX2 carrier)

- 4 Cortex-A57 cores (plus 4 A53 cores, unused)
- 256-core “Maxwell” GPU
- 4GB LPDDR4 RAM
- 16GB eMMC
- Ethernet MAC
- WiFi/Bluetooth module
- On-chip audio, video, image processing engines
- SPI, I2C, HDMI, MIPI DSI and CSI, GPIOs, PCIE, SDIO, SATA, USB 2 & 3

Jetson TX2



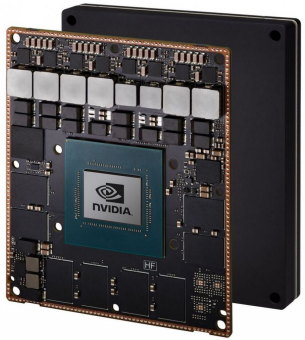


Introduced in 2017, 4GB and TX2i models in 2018

- 4 Cortex-A57 cores plus 2 “Denver2” cores
- 256-core “Pascal” GPU
- 4/8GB LPDDR4 RAM
- 16/32GB eMMC
- Ethernet MAC
- WiFi/Bluetooth module (on TX2 only)
- On-chip audio, video, image processing, and sensor processing engines
- SPI, I2C, HDMI, MIPI DSI and CSI, GPIOs, PCIE, SDIO, SATA, USB 2 & 3
- TX2i for industrial temperature ranges

- SDcard slot, SATA connector
- DSI/eDP/DP/HDMI
- PCIe x4 slot, M.2 Key E
- OV5697 camera
- Heatsink + fan
- USB OTG and USB 3.0, Ethernet
- I/O headers
- Power supply

Jetson AGX Xavier



Introduced in 2018
High end of the line
AI/ML focused

- 8 “Carmel” cores (ARMv8.2A)
- 512-core “Volta” GPU with 64 Tensor Cores
- 2 deep-learning accelerators
- 32GB LPDDR4 RAM
- 32GB eMMC
- Ethernet MAC
- On-chip audio, video, image processing, and sensor processing engines
- SPI, I2C, HDMI, MIPI DSI and CSI, GPIOs, PCIE, SDIO, SATA, USB 2 & 3

- MicroSD/UFS combo, USB/eSATA connector
- 2xUSB type C, 1 USB micro-B
- PCIe, M.2 Key E, M.2 Key M
- Ethernet
- I/O and camera headers
- Power supply

Jetson Nano

Introduced in 2019
(2GB model in 2020)

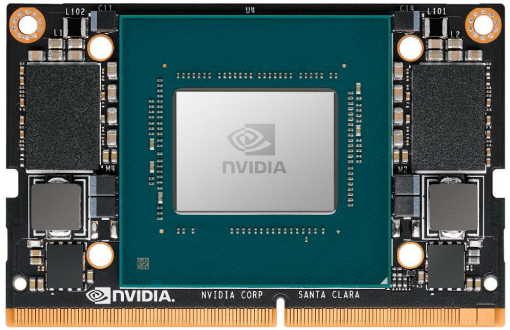
Entry-level/hobbyist market

Some compatibility with Raspberry Pi peripherals

- 4 ARM Cortex-A57 cores
- 128-core “Maxwell” GPU
- 4GB LPDDR4 RAM
- 16GB eMMC (or SDcard slot)
- Gig Ethernet MAC
- On-chip audio, video, image processing engines
- SPI, I2C, HDMI, MIPI DSI and CSI, GPIOs, PCIE, SDIO, SATA, USB 2 & 3

- USB OTG (w/power)
- 4x USB 3.0 type A
- HDMI and DP
- Ethernet
- Jack for external power supply
- MIPI CSI connectors
- M.2 Key E
- I/O headers
- Fan connector

Jetson Xavier NX



Introduced in 2020

Same form factor as Jetson Nano

"Replacement" for TX2

- 6 "Carmel" cores (ARMv8.2A)
- 384-core "Volta" GPU with 48 Tensor Cores
- 2 deep-learning accelerators
- 8GB LPDDR4 RAM
- 16GB eMMC
- Ethernet MAC
- On-chip audio, video, image processing, and sensor processing engines
- SPI, I2C, HDMI, MIPI DSI and CSI, GPIOs, PCIE, SDIO, SATA, USB 2 & 3

- USB micro-B (device)
- 4x USB 3.1 type A (host)
- HDMI and DP
- MIPI CSI connectors
- M.2 Key E, Key M
- Ethernet, WiFi/BT module
- I/O headers
- Fan connector
- Power supply

Vendor-provided software

Visit https://developer.nvidia.com/embedded/develop/software
for more information

BSP - “Linux4Tegra”

- Linux kernel (NVIDIA downstream)
- Bootloaders (NVIDIA proprietary and U-Boot)
- Firmware
- Drivers and hardware-specific libraries
- Power management
- Userland graphics and multimedia support
  - Libdrm shim, GL/EGL/GLES, Vulkan loader, X.org driver, v4l2 and gstreamer plugins
- Configuration files
- Tools for flashing
- Ubuntu 18.04-based “sample” root filesystem

JetPack - SDKs for application development

- CUDA tools and libraries
- NVIDIA Nsight
- CuDNN
- VisionWorks and VPI
- TensorRT
- Jetson Multimedia API
- DeepStream SDK

Imaging and flashing support in L4T

- **Devices are initially flashed via USB**
  - NVIDIA custom protocol and binary-only tools
  - Formats the on-board storage and programs all partitions
- **L4T provides a `flash.sh` script to wrap the imaging and flashing tools**
  - Queries the device for model/version/revision info via USB
  - Creates rootfs, formats/signs the boot components
  - Loads a flasher/recovery image over USB
  - Sends all components via USB for flasher to program

The meta-tegra layer

Visit https://github.com/OE4T/meta-tegra for more information

Genesis of meta-tegra

- Began in 2015 (OE-Core ‘jethro’, L4T R23.1) for Jetson-TX1
- Integrated into an existing distro
- Needed hardware-specific features enabled by NVIDIA binary blobs, including CUDA
- Initial development with Jetson-TX1 development kit, but needed to support custom product based on the module

What a BSP layer provides

- **Basics for booting the board**
  - Kernel, bootloader(s), drivers, firmware
- **Machine configuration files**
- **Any other board specifics**
  - Image creation (via image types, wic, etc.)
  - Board-specific config files (ALSA, Bluetooth, network ,etc.)
- **Recipes for board-specific tools and examples**
  - Such as diagnostics/manufacturing tools

Kernel recipe in meta-tegra

- **Vendor-provided downstream kernel**
  - Available via git, spread across 15 (!) separate repositories
  - 4.9 base plus Android and vendor-specific patches
  - Used git subtrees to re-integrate into a single repo
- **Minimal additional patches**
  - Mostly for compiling with newer toolchains
  - With a few bugfixes
- **Uses in-tree defconfigs and `linux-yocto.bbclass`**
  - Originally used per-machine defconfig files with recipe

Bootloader recipes in meta-tegra

- **Complex boot sequence with multiple stages of bootloaders (varies by SoC) and Trusted OS**
- **Some boot components are binary-only**
- **TX1/Nano and TX2 support U-Boot**
  - Optional on TX2, no U-Boot on Xavier
  - U-Boot patches mostly upstreamed now
  - All of the heavy lifting is done by `cboot`
- **Sources for cboot sometimes available**

Machine configurations in meta-tegra

- Configurations for all current development kits are available (and some non-devkit modules)
- Using `SOC_FAMILY` support for SoC-specific recipes/packages
- Additional machine overrides for `tegra`, `cuda`
- Added `TEGRA_PKGARCH` for generic (but Tegra-specific) packages

Imaging and flashing tools in meta-tegra

- **Added an image types bbclass for creating a `tegraflash` package**
  - Includes everything needed to flash the target - scripts, tools, and all target components
- **Flash and SDcard helper scripts to wrap the lower-level tools**
- **Additional handling for bootloader update payloads**
- **Hooks for signing boot components during the build**

Firmware, configuration files, etc.

- **Recipes to extract other runtime components from the BSP package**
  - Power model daemon and profiles
  - Power Hinting Service
  - “Driver” libraries
  - Argus (camera/ISP control) daemon and libraries
  - Firmware loaded by the kernel (BT/Wifi, GPU, etc.)
- **Initscripts/systemd service units**
  - Just those needed by hardware

Graphics support

- **SoC combines CPU and GPU**
- **GPU driver provided in source form with kernel**
- **Some userland components are binary-only**
  - `libdrm.so` shim (no DRM/DRI in kernel)
  - OpenGL/EGL/GLES vendor library for use with `libglvnd`
  - Xorg driver
- **Wayland support via egl-wayland**
  - Weston with eglstreams patches
- **Vulkan loader**

Multimedia support

- **Plugins for libv4l2 and gstreamer1.0**
  - Mostly sources in recent versions
  - Interfaces to hardware video encoder/decoders, CUDA, ISP
  - Output to EGL surfaces
  - Handling of NVMM buffers
- **Jetson-specific multimedia APIs**
  - “SDK” provided (headers and sample code)

JetPack components

- **Mostly simple extraction/repackaging of files from NVIDIA's `deb` packages**
- **CUDA is more complicated**
  - Added `cuda.bbclass` for configuring and cross-compiling CUDA applications
  - Host-side tools must be pre-downloaded
  - Modifications for cmake
  - SDK support
  - Toolchain version dependencies

Virtualization support

- **Added in L4T R32.x**
- **Jetson-specific container runtime/tools**
  - Pass-through container mounts for many of the userland libraries in the BSP
- **NVIDIA provides Jetson-specific Docker containers downloadable from NGC**

The meta-tegra/contrib layer

- Software layer to support meta-tegra builds
- Includes recipes for older gcc versions (for CUDA)
- Recently added gstreamer 1.14 recipes (for containers and Deepstream)

The Challenges of maintaining a BSP layer

What has gone well

- Using the layer for real products
- Keeping up with OE-Core/Yocto Project releases
- Keeping configurations close to L4T stock settings
- Test distros with automated builds
- Having supportive employers

Lessons learned

- **There are other machines, architectures, etc. out there**
  - Use `PACKAGE_ARCH`, `COMPATIBLE_MACHINE`
  - Be careful about modifying common recipes
- **Put machine-specific settings in machine `.conf` files**
  - Overrides should be used judiciously
- **Minimize layer dependencies**
- **Use existing infrastructure when you can**
  - For u-boot builds, kernel config handling
- **Your build/OS configuration/workflow isn't the only one**
  - Initscripts vs. systemd, rpm/deb/ipk, SDK builds, etc.

Challenges

- **Complexity of the platform**
- **L4T release inconsistencies and quality issues**
- **Compatibility issues (past, present, future):**
  - Downstream old kernel with no upstream option
  - Binary-only blobs in graphics stack and elsewhere
  - Toolchain version issues
- **BSP vs. application support**

Conclusion

What's next

- **Grow the community**
  - More active developers needed
  - Contribution guidelines, CoC, issue templates, etc.
  - Communication channels
- **More documentation**
  - Wiki, How-To docs, technical details
- **Add content**
- **Make it easier to get started**
  - Demo/reference distro

Contact info

- GitHub: https://github.com/OE4T
- Slack: https://oe4t.slack/com (visit https://github.com/OE4T/meta-tegra/wiki for link to join)
- E-mail: matt@madison.systems

Thanks for your time

yocto PROJECT | THE LINUX FOUNDATION